Sequential Logic Design

- Design the following circuits using Verilog **and create a testbench** for each design to check its functionality. **Create a do file for question 3.**

- Testbenches are advised to be a mix between randomization and directed testing taken into consideration realistic operation for the inputs. Apply self-checking condition in the testbench for at least one of the design below.

1) Implement the following latch as specified below

Parameters

LAT_WIDTH: Determine the width of input data and output q

Ports

Name	Type	Description
aset	Input	Asynchronous set input. Sets q[] output to 1.
data[]		Data Input to the D-type latch with width LAT_WIDTH
gate		Latch enable input
aclr		Asynchronous clear input. Sets q[] output to 0.
q[]	Output	Data output from the latch with with LAT_WIDTH

If both aset and aclr are both asserted, aclr is dominant. aset signals sets **all** output bits to 1

2) Create a hand-drawn schematic for the following Verilog design

1-

```
module reg_blocking1(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q1 = 0;
        q2 = 0;
        out = 0;
    end
    else begin
        q1 = in;
        q2 = q1;
        out = q2;
    end
end

endmodule
```

```
module reg_non_blocking1(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q1 <= 0;
        q2 <= 0;
        out <= 0;
    end
    else begin
        q1 <= in;
        q2 <= q1;
        out <= q2;
    end
end

endmodule
```

```
module reg_blocking2(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q2 = 0;
        q1 = 0;
        out = 0;
    end
    else begin
        q2 = q1;
        q1 = in;
        out = q2;
    end
end

endmodule
```

```
module reg_non_blocking2(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q2 <= 0;
        q1 <= 0;
        out <= 0;
    end
    else begin
        q2 <= q1;
        q1 <= in;
        out <= q2;
    end
end

endmodule
```

```
module reg_blocking3(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        out = 0;
        q2 = 0;
        q1 = 0;
    end
    else begin
        out = q2;
        q2 = q1;
        q1 = in;
    end
end

endmodule
```

```
module reg_non_blocking3(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        out <= 0;
        q2 <= 0;
        q1 <= 0;
    end
    else begin
        out <= q2;
        q2 <= q1;
        q1 <= in;
    end
end

endmodule
```

2-

```
module comb_blocking1(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  x = a & b;
  y = x | c;
end

endmodule
```

```
module comb_non_blocking1(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  x <= a & b;
  y <= x | c;
end

endmodule
```

```
module comb_blocking2(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  y = x | c;
  x = a & b;
end

endmodule
```

```
module comb_non_blocking2(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  y <= x | c;
  x <= a & b;
end

endmodule
```

```
module comb_non_blocking3(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;
reg y_comb;

always @(*) begin
  x = a & b;
  y_comb = x | c;
end

always @(posedge clk) begin
  y <= y_comb;
end

endmodule
```

3)

1. Implement 4-bit counter with asynchronous active low set using behavioral modelling that increment from 0 to 15
   - Input:
     - clk
     - set (sets all bits to 1)
     - out (4-bits)
2. Use the structural counter done in Assignment3_v13 as a golden reference.
3. Test the above behavioral design using a self-checking testbench
   - Testbench should instantiate both structural and behavioral counters.

4) Extend on the previous counter done in question 3 to have extra 2 single bit outputs (div_2 and div_4). Hint: Observe the output bits of the "out" bus to generate the following
   - div_2: output signal that acts as output clock with a frequency half the input clock signal.
   - div_4: output signal that acts as output clock with a frequency quarter the input clock signal.

   Note that your document should be organized as 4 sections corresponding to each design above, and in each section, I am expecting the Verilog code, and the waveforms snippets